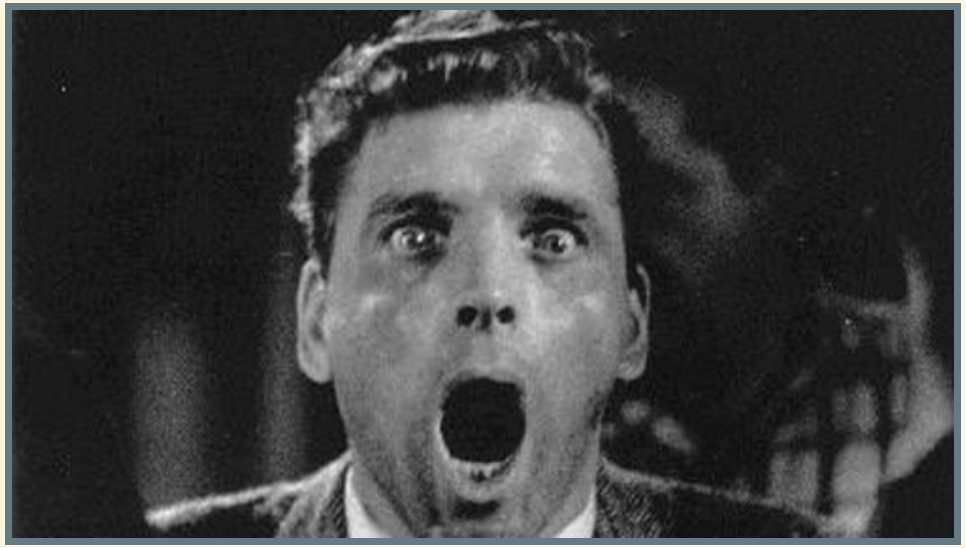
FROM STUPID TO SOLID CODE!

A few basic principles of Object-Oriented Programming and Design.

DISCLAIMER

THESE ARE PRINCIPLES, NOT LAWS!



STUPID CODE, SERIOUSLY?

WHAT MAKES CODE STUPID?

- Singleton
- Tight Coupling
- Untestability
- Premature Optimization
- Indescriptive Naming
- Duplication

SINGLETON

Programs using global state are very difficult to test.

Programs that rely on global state hide their dependencies.

Why Singletons Are Controversial
Why is Singleton considered an anti pattern?
So Singletons are bad, then what?

TIGHT COUPLING

Generalization of the **Singleton** issue.

Also known as **strong coupling**.

Reducing Coupling (Martin Fowler)

UNTESTABILITY

Testing should not be hard!

Whenever you don't write **unit tests** because you **don't have time**, the real issue is that your code is bad.

PREMATURE OPTIMIZATION

Premature optimization is the root of all evil. — Donald Knuth

There is **only cost** and **no benefit**.

PrematureOptimization Anti-Pattern

INDESCRIPTIVE NAMING

Name your classes, methods, attributes, and variables properly.

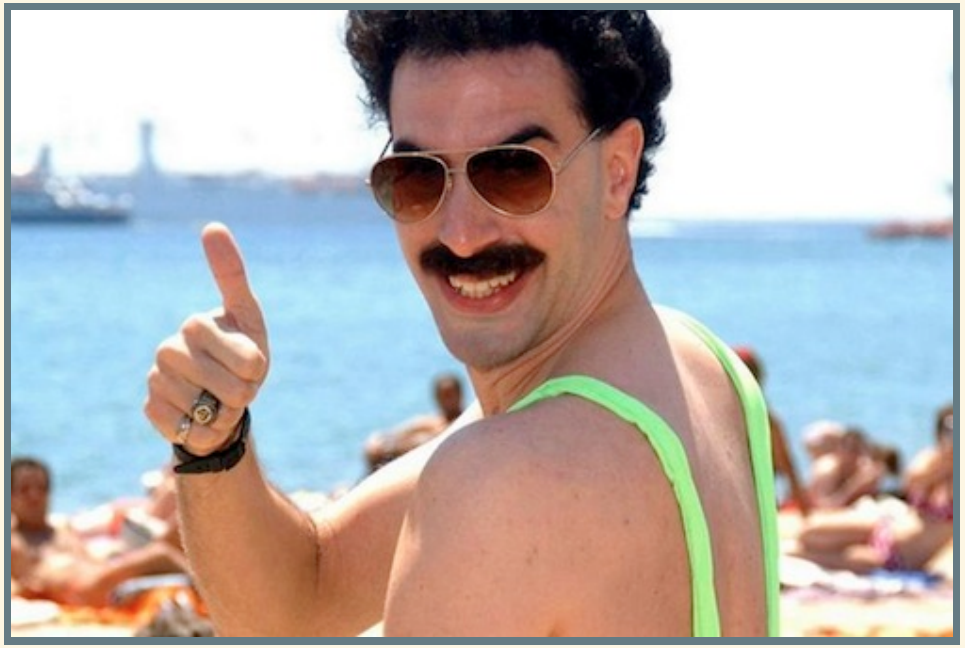
Don't abbreviate, never!

DUPLICATION

Don't Repeat Yourself!

Keep It Simple, Stupid!


KEEP
CALM
AND BE
AWESOME
INSTEAD



SOLID TO THE RESCUE!

SOLID

A term describing a collection of design principles for **good code** that was coined by **Robert C. Martin** also known as **Uncle Bob**.

SOLID

SINGLE RESPONSIBILITY PRINCIPLE (SRP)

There should **never** be **more than one reason**
for a class to change.

The Single Responsibility Principle

Just because you can, doesn't mean you should!

#PROTIPS

- Split big classes
- Use **layers**
- Avoid **god** classes
- Write straightforward comments

SOLID

OPEN/CLOSED PRINCIPLE (OCP)

Software entities should be **open for extension**,
but **closed for modification**.

The Open-Closed Principle

OPEN CLOSED PRINCIPLE

Open Chest Surgery Is Not Needed When Putting On A Coat

#PROTIPS

- Make all member variables **private**
- No global variables, ever
- Avoid **setters** (as much as possible)

SOLID

LISKOV SUBSTITUTION PRINCIPLE (LSP)

Objects in a program should be replaceable with instances of their subtypes **without altering the correctness** of the program.

Liskov Substitution Principle
The Liskov Substitution Principle

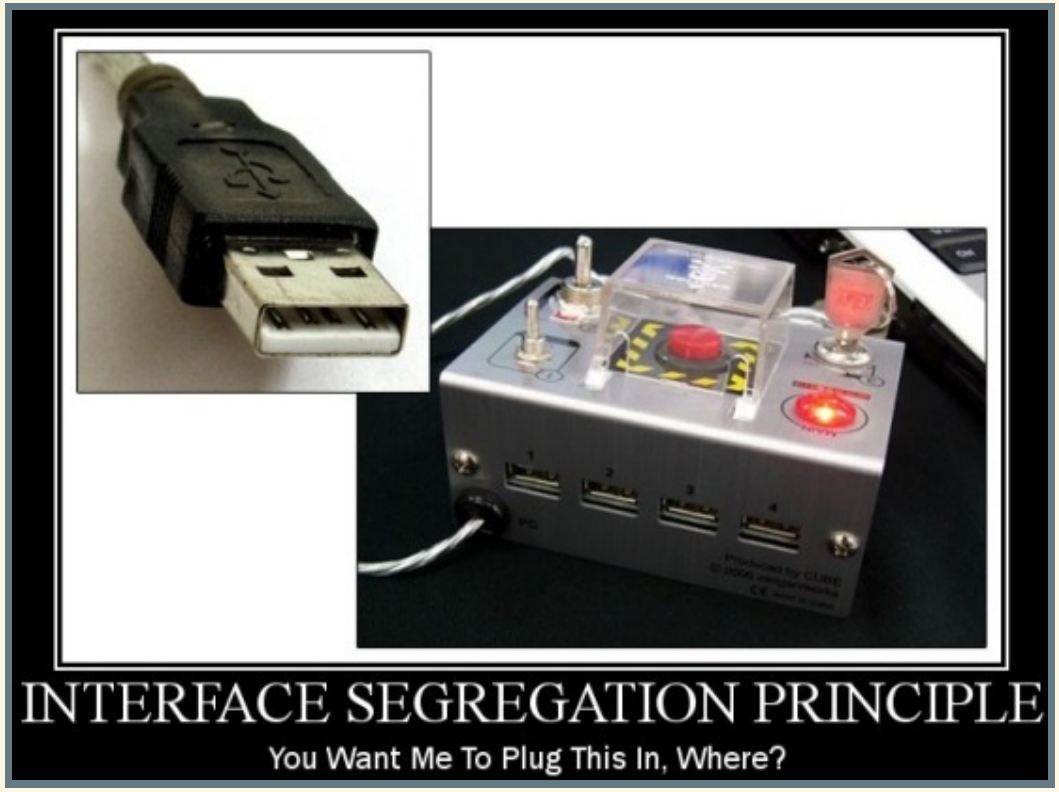

LISKOV SUBSTITUTION PRINCIPLE
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction


SOLID

INTERFACE SEGREGATION PRINCIPLE (ISP)

Many client-**specific** interfaces are **better than one** general-purpose interface.

The Interface Segregation Principle


INTERFACE SEGREGATION PRINCIPLE
You Want Me To Plug This In, Where?

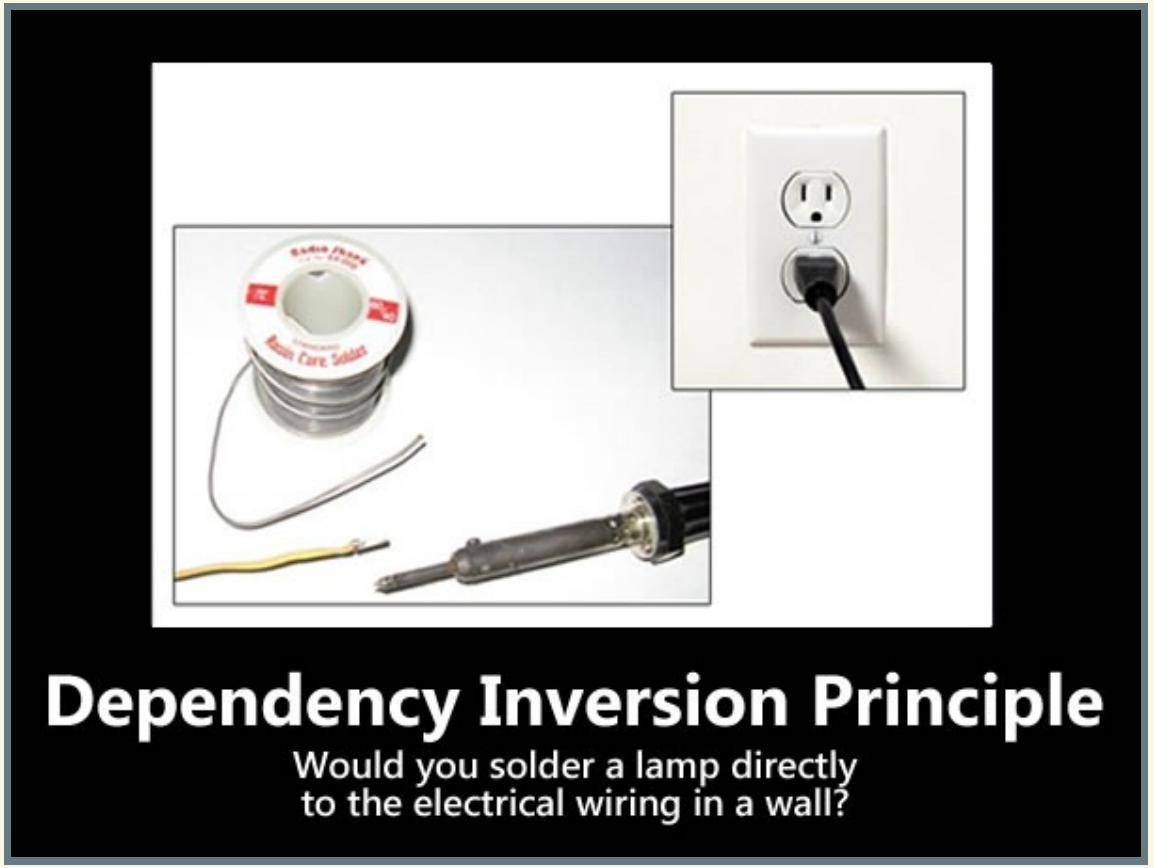
SOLID

DEPENDENCY INVERSION PRINCIPLE (DIP)

High level modules **should not depend upon low level** modules.
Both **should depend upon abstractions**.

Abstractions should not depend upon details. Details should depend upon abstractions.

DIP in the Wild
Dependency Inversion Principle
The Dependency Inversion Principle
Dependency Injection Is NOT The Same As The Dependency Inversion Principle


Dependency Inversion Principle
Would you solder a lamp directly
to the electrical wiring in a wall?

CONCLUSION

Avoiding **tight coupling** is the **key**!

Use your brain.

Writing SOLID code is not that hard.

THANK YOU, QUESTIONS?

williamdurand.fr
github.com/willdurand
twitter.com/couac

CREDITS

http://lostechies.com/derickbailey/2009/02/11/solid-development-principles-in-motivational-pictures/